View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DCU Online Research Access Service

Noname manuscript No.
(will be inserted by the editor)

An Extended Ontology-based Context Model and Manipulation Calculus for Dynamic Web Service Processes

Kosala Yapa Bandara · MingXue Wang · Claus Pahl

Received: date / Accepted: date

Abstract Services are offered in an execution context that is determined by how a provider provisions the service and how the user consumes it. The need for more flexibility requires the provisioning and consumption aspects to be addressed at runtime. We propose an ontology-based context model providing a framework for service provisioning and consumption aspects and techniques for managing context constraints for Web service processes where dynamic context concerns can be monitored and validated at service process run-time.

We discuss the contextualization of dynamically relevant aspects of Web service processes as our main goal, i.e. capture aspects in an extended context model. The technical contributions of this paper are a context model ontology for dynamic service contexts and an operator calculus for integrated and coherent context manipulation, composition and reasoning. The context model ontology formalizes dynamic aspects of Web services and facilitates reasoning. We present the context ontology in terms of four core dimensions – functional, QoS, domain and platform – which are internally interconnected.

Keywords Dynamic aspect · Context model ontology · Context constraints · Context manipulation · Service process

1 Introduction

The execution of a Web service is determined by how it is provisioned by the provider in terms of functional and non-functional properties, but also how a service is consumed by the user in the context of technical and business-level settings and requirements. These concerns define the execution context for a service or a service process that range from interfaces to quality to business settings like governance and domain aspects to platform, communication and devices. Our concerns here are context aspects of relevance for the execution. Service-centric applications need management in the form of monitoring and validation at run-time because of new versions of selected services or new services supplied by different vendors, different execution time contexts hamper the correctness and quality levels of Web service applications with respect to their contextual expectations. Traditionally, applications are validated before their deployment [37], but monitoring and validation at run-time is needed to address flexibility requirements [55,9,8].

The notion of context is extensively investigated in mobile and pervasive applications to define locative and temporal aspects in dynamic applications [28,33, 50]. CONON [56] and SOUPA [25] are widely used context models in pervasive computing environments. They address fundamental context aspects such as device, location, person and activity for capturing information about the execution situation. While these context models do not characterize dynamic aspects of Web services as software entities embedded into business processes, their formal context representation and knowledge sharing and reasoning aspects provides some input to our research. The notion of context can be used to define functional and non-functional features of Web services [37,38], there focusing on context matching for service selection, but only statically for the design stage. Rosemann et al. [47] have focused on context models in business processes and proposed a conceptual con-

Kosala Yapa Bandara · MingXue Wang · Claus Pahl
School of Computing,
Dublin City University, Dublin, Ireland.

text taxonomy, but acknowledge the need for further research on process execution. We follow Truong and Dustdar [52] here in defining our encompassing context notion, who consider context information as any additional information that can be used to improve the behaviour of a service in a situation. They observe that "while some types of context information, such as location, presence, individual profile, machine/device and network, have been widely used in many context-aware systems for a long time, other types of context information, such as service/application, activity/task, and team, are also considered in web service context-aware systems". Our context notion will capture classical functional and non-functional aspects (as service-based properties determined by the provider) and also domain and platform aspects (as abstract and concrete properties determined by the consumer).

We can identify the following gaps in the literature:

- The available context categorizations and models do not sufficiently describe and integrate dynamic service context. A complete context model ontology to conceptualize dynamic service context is needed.
- The contextualization of Web service processes, i.e. the definition of a context model for service processes, is required to support validation monitoring of dynamic requirements at process run-time.
- Dynamic requirements can be defined as context constraints, and need to be supported by context reasoning features of the ontology.
- The manipulation and reasoning of dynamic service context specifications is necessary for dynamic requirements.
- The available constraints instrumentation and validation monitoring approaches do not sufficiently address run-time instrumentation and validation monitoring of dynamic requirements.

Addressing the first three directly, i.e. modelling dynamic context aspects in service processes, is our focus, aiming to provide a complete model that contextualises service processes. A context model and an operator calculus are our contributions. The purpose of such a context model is to support a context-aware approach to manage dynamic requirements in a service process at runtime, based on an identification of dynamic service context aspects and their formalisation in the model and calculus, i.e. to contribute to the remaining challenges. Requirements that can be changed at process run-time (aspects that vary for individual service processes), such as cost of a service, security needs, process runtime aspects or payment aspects are *dynamic requirements*. Dynamic requirements arising from the context model can be operationalised as context constraints. This contribution can be utilised to generate context constraints and allow their instrumentation and validation at runtime [37,38].

The novelty of our contribution is a context framework to model dynamic, operational aspects for a service process at runtime based on a semantic model of context that deals with diverse, but integrated functional and non-functional dynamic aspects of Web service processes. This context model provides a classification and formalisation of dynamic aspects. It works as a conceptualisation for Web service processes that specifically allows interdependencies between model aspects to be determined, specified, manipulated and reasoned about. This semantic model is embedded into a rich conceptual modelling technique for dynamic service contexts including language and operator calculus elements. The ontology framework with its operator support for context manipulation and composition goes beyond normal ontology models.

The remainder of this paper is organized as follows. In Section 2, we motivate our research using scenarios and concrete examples. In Section 3, a conceptual context model is developed focusing on dynamic aspects relevant for composition and execution of Web service processes. In Section 4, the conceptual model is formalised as a context model ontology. Current formalisation techniques are discussed and ontology-based context modelling and formalisation are detailed. Section 5 addresses techniques for context manipulation and composition. In Section 6, the context model is applied, illustrating context constraints and discussing context operationalisation. In Section 7, we discuss related work. Finally, we conclude our contribution.

2 Motivation

2.1 The need for service process contextualisation

Dynamic service context aspects contribute to effective composition and coordination [3]. Service matching and service selection approaches support process design-time validation for service-based applications [37]. The effective composition and coordination at process run-time needs to involve service management techniques. Here are some motivating examples for changing requirements in a dynamic service context.

- Service response time is a constraint – the service response time cannot be pre-defined; it varies. That is, service response time is a dynamic aspect, which is needed for effective composition and collaboration at process run-time, controlled by the provider.
- Cost of a service process is a constraint – if a service fails at run-time, then a new service should replace it

without violating a cost constraint. Cost of a service can also be changed based on currency exchange rates, which arise from the business domain of the consumer.
- A service can be executed on selected devices – service execution can depend on device features; that is, device context is needed for effective composition and collaboration of Web services.
- A service needs to be adapted, depending on dynamically changing consumer locations (and, in a wider sense, locales as aggregations of lingual, location and regulatory settings ranging from units to currencies or taxes).

These relate to functionality and quality of service provided, and platform and domain aspects (environmental aspects at execution time) such as execution engine, network/platform services, domain ontologies and standards.

Web services enable business processes to be more dynamic and flexible, providing more integration support. Some aspects such as response time, availability, reliability and also some business constraints, can only be guaranteed at process runtime, as discussed above. Service level agreements (SLAs) defined between parties need to be monitored. A change of a dynamic aspect may affect on other aspects, e.g. a client may need a Web service process with low response time or high security. In dynamic service applications, heterogeneous services need to be combined at process run-time based on various dynamic requirements such as user location, language needs, etc.

This discussion shows that Web service processes need dynamic service context instrumentation and validation. Therefore, a Web service process needs to be contextualized, i.e. be made context-aware to monitor dynamic requirements at process run-time. The term contextualisation refers here to integrating operational aspects in the execution space, i.e.,

- those determined by the provider (functional and non-functional quality aspects of the provisioning of the service) as well as
- those determined by the consumers (in more abstract terms the domain and in more concrete terms the platform on which the service is consumed)

2.2 Use case

We choose a use case to clarify our context notion. We use an environment that provides service-level access to stock market information and analyses[1]. A German user might want to access data from the New York stock exchange, which is provided in an English format. We present a scenario in which the service consumer can implement a context-dependent interface, i.e. one that allows technical interaction of service interface and description aspects in German (as the language) and from a German regulatory context (currencies, units and taxes) as specific aspects.

At the application-level, two sample calls of a stock market data analysis service for two locales[2] (US-locale with English as the language and USD as currency and DE-locale with German as the language and EUR as the currency) could be: *Analyse(10/30/2011, logistics) → 3.82 USD* and *Analysis(30.10.2011, Logistik) → 4.23 EUR*. Context-depending artefacts for service processing resulting from the consumption context of the user in this example are:

- Date: a format change is needed – which would also apply to time and collation issues,
- Sector: data values describing an industry sector are localised based on a translation between standardised terminologies – which would also apply to product categories,
- Language: operation names (and possibly other interface and model elements) are translated between languages,
- Currency: values are converted – as would be other measurements and units.

This list can be extended: different regulatory environments based on maybe multilingual and standardised glossaries and dictionaries; calculations and conversions based on rules (fixed) or repositories (dynamic); tax rates and customs duties can be added if products are sold; any messages including help and error messages to which text translation would be applied. Typical examples for technical terms that need translation or mappings as forms of adaptation in the banking or stock markets context are (average price - Durchschnittspreis), (main trading phase - Haupthandelsphase), or (volume weighted average - volumengewichteter Durchschnitt) that are based on accepted, often standardised terminologies. Some examples might be defined in terms of classification and categorisation standards: (logistics - Logistik) for a sector or (dairy - Milchprodukte) for product categories. An observation is that a range of

[1] This is based on a case study using financial stock market information services from http://xignite.com/ and http://deutsche-boerse.com.

[2] The term locale combines here context aspects that define the domain context (abstract settings) of the consumer. We have singled out this domain context as it bridges between the more commonly used quality and platform aspects.

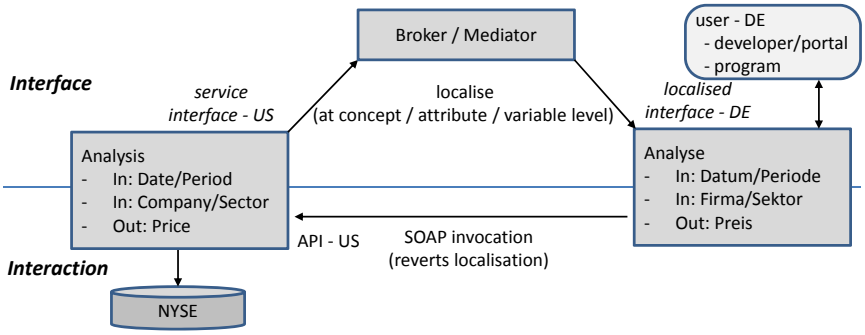

Fig. 1 Contextualisation of Stock Market Analysis Feature - Focus on Service API.

context aspects are interdependent: the language might determine variants of a standard being used. The regulatory settings in terms of units might affect the functionality and interface of the services being used.

In this scenario, the consumer context settings and requirements differ from the provider context assumptions. A context model should provide a list of relevant, possibly differing concerns. A mediator can provide automated adaptations. An operationalisation of the context model can then result in context constraints to be dynamically generated and monitored through probes.

In the example, the user-DE can discover services based on a German specification and can invoke them based on a German interface. A stock market analysis provider can add a DE-context to its default US-context. This would result in a correct match in a full negotiation process in which a user searches for services that are provided in a context-specific way since the provider is able to support US-to-DE locale mappings if required. In an architecture that implements these mappings and translations, service instrumentation would result in a process to be generated and enacted, rather than a single SOAP request as indicated in Fig. 1. This process could comprise service invocation and logging (location) for accountability where the location is a parameter, which indicates where and how records are kept (if ruled by privacy laws). The above scenario could be further extended to allow an American user (locale US) to access a German-language stock market information provider, e.g. Deutsche Börse, Frankfurt.

Match-making between provider specifications and consumer requirements, as it would happen in SLA negotiations, is not our concern. Neither is the automated adaptation or monitoring which could result from the scenario described. We focus here on a comprehensive conceptual framework and incorporate aspects that allow coherent manipulation and reasoning of a context model for dynamic service processes to take place. The case study has illustrated the need for a coherent and inclusive process-oriented context model and calculus.

3 Context Model Development

3.1 Context for dynamic services

A notion of context requires more than the current widely accepted building blocks of the Web service description. However, there is no widely accepted definition for context in information science. Context is defined and used in various applications in their own perspectives [36,21,18], in particular to define locative and temporal aspects in mobile [50] and ubiquitous system [28,33] applications. Service composition can be static or dynamic. In static composition, services to be composed are selected at process design-time. In dynamic service composition, services to be composed are selected at process run-time [20]. In previous work on service composition, context has been explored for service discovery and selection at process design-time [37]. However, there is still a gap where context for dynamic services and context operationalization are needed at service process run-time in order to validate dynamic requirements. Therefore, the context notion needs to be rich enough to illustrate dynamic aspects relevant to composition and execution of Web services. In order to address these needs, we define **context for dynamic services** or **dynamic service context** as follows.

Dynamic service context is client, provider or service related information, which enables or enhances effective composition and collaboration between them.

Consequently, the explicit formalisation of dynamic aspects relevant to composition and execution of Web services into a processable model (context model) is the central objective.

Truong and Dustdar [52] see **context-aware web services** are a subtype of context-aware systems. They consider context information as any additional information that can improve the behaviour of a service in a situation. Without such additional information, the service might operate normally, but with context information, the service can operate better. They observe

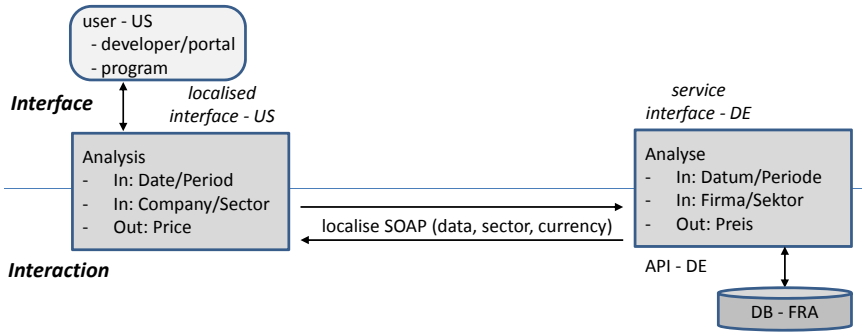

Fig. 2 Contextualisation of Stock Market Analysis Feature - extends NYSE-specific Analysis Service to include DB-FRA.

that while some types of context information, such as location, presence, profiles, devices and network, have been widely used in many context-aware systems, other types of context information, such as service functionalities and activities, but also stakeholders, are in fact also considered in service context notions. We follow this broad context notion, but focus in the light of our dynamic concerns on an operational context. Raik et al. [45] consider these dynamic features offered by the framework in a shared context model, describing the operational environment of the system. The context is defined through a set of context properties.

Service provisioning and consumption frame the context notion here. Provisioning is the process of preparing and equipping IT infrastructure to allow it to provide (new) services to its users. Service provisioning languages, such as SPML, support exchanging user, resource and service provisioning information between cooperating organizations. SPML is an open standard for the integration and interoperation of service provisioning requests, but deals more with the management of resources. As we are concerned with higher-level aspects, the context notion is more appropriate. A service-level agreement (SLA) is a part of a service contract where a service is formally defined. An SLA will typically have a technical definition in terms measurable details. Although there are similarities, we see the context spaces as a framework in which SLA negotiation and specification can take place, and also dynamic adaptations, as illustrated with the use case, can take place.

3.2 Context model determination

We followed a systematic approach to elicit and define dynamic service context aspects for our context model. A general and complete context taxonomy is important for context-aware dynamic Web service applications. We discuss the taxonomy development methods, which includes empirical experiments before detailing context model ontology development.

Our context taxonomy development methodology has two steps. Step 1 involves two parts. They are the analysis of real world application scenarios and the analysis of context classifications in the literature for capturing dynamic service context, which we defined in section 3.1. Step 2 involves context orientation where we initially follow two more general perspectives, then we further detail the orientation of context categories, based on a range of criteria.

Step 1. This step involves two parts where we focused on capturing all possible context categories relevant to dynamic service context:

- In the first part, we used an empirical analysis of application scenarios in a classical business domain. Scenarios from commercial applications of different system architectures were considered with the help of domain experts. We explored dynamic aspects of constituent services relevant to application scenarios focusing on service composition and execution at service process run-time.
- In the second part, we considered domain-specific context taxonomies, comprehensive business services, and process context models, particularly as described in [47, 27]. We captured dynamic aspects, taking the perspective of Web services in general and focusing on service composition and execution at process run-time. Most of previous work is domain-specific, such as [56, 48, 13]. However, the community structure proposed in [38] is more general than other approaches and we adapted some aspects such as run-time attributes, business attributes, and security attributes from it.

Step 2. The organisation of context attributes in a general context taxonomy is important in the literature.

- We separated the identified context categories into **inward** and **outward** perspectives on Web services.

In the outward perspective, dynamic aspects relevant to service interfaces and quality of service properties were captured – the provisioning view controlled by the provider. In the inward perspective, dynamic aspects relevant to process execution environment stemming from the consumption by the user were identified. We further describe these perspectives in section 3.3.
- We then classified context categories and subcategories having different criteria until the taxonomy becomes more general. This detailed classification was supported by the literature related to various non-functional and context classifications, such as [14,38,37,56].

Step 1 and Step 2 were iteratively followed until the context model becomes complete for a more general perspective, i.e. we did not observe further changes based on the application scenarios chosen. Our observations led to the development of a flexible and evolvable context model focusing on dynamic aspects of Web services and Web service business processes [41,4].

The focus of the empirical determination and evaluation was validity and completeness of context categories; categories defined in the context ontology must represent the needs of dynamic requirements (validity) and all the required dynamic requirements need to be covered (completeness). The evaluation process involved application scenarios from two complementary domains and expert opinion analysis. Application scenarios from a classical business domain were analyzed during the development of the semantic context model. We followed a formative evaluation approach to evaluate the context model. The following two complementary domains were considered:

- content-oriented domain, in particular courseware generation for e-learning applications,
- convenience services domain, in particular a technical tool support service.

The definition and analysis of these application scenarios were supported by the domain experts. These scenarios were developed focusing on real-world business applications. Expert opinions were collected to analyse the validity and completeness aspects of context categories and dynamic service context definition using an online questionnaire. Based on the results from the formative evaluation and expert opinion analysis, we incorporated the initial context model with minor adjustments and considered the resulting context model is stable.

3.3 Core context model definition

Our context model focuses on dynamic requirements relevant for service processes at runtime. This is at this stage a core model defining a vocabulary, leaving concrete values uninterpreted. We define context model as a specification,

$$Context\ Model = \langle \Sigma, \Phi \rangle$$

with

- a signature $\Sigma = \langle C, R \rangle$ consisting of **concepts** C and **roles** R to define context aspects and their attributes.
- **context descriptions** $\phi \in \Phi$ based on Σ. $\Phi = \langle C \leftrightarrow R \rangle$ defines properties in terms of concepts and roles as description logic formulas (as a formal foundation of an ontology language).

Moreover, the mechanisms for modifying and composition context descriptions are an important part of the overall model, which will be addressed in Section 5.

The context model taxonomy is shown in Figure 3. Central are four core aspects under which specific aspects are captured. These *core aspects* represent fundamental dimensions of context relevant to Web service composition and execution.

- *Outward (provisioning)*: two of them are linked to how a service interacts with and impacts on its environment: the *functional context* captures the functional capabilities from an input/output and precondition/post-condition perspective and the *quality context* captures non-functional aspects at the service interface.
- *Inward (consumption)*: the other two capture how the user and deployment environment impact on service execution: the *domain context* captures dynamic requirements stemming from the application domain of the service and the *platform context* captures dynamic requirements stemming from its technological environment.

Where possible, these context categories were aligned with standardised or widely used vocabularies, such as software quality standards (ISO 9126) or business directory information (UDDI) for the quality context.

Functional context describes the operational features of services.

- Syntax: includes the input/output parameters that define messages of operations and the data types (or semantics) of these parameters for service invocation.
- Effect: includes the pre- and post-conditions, i.e. the operational effect of an operation execution.

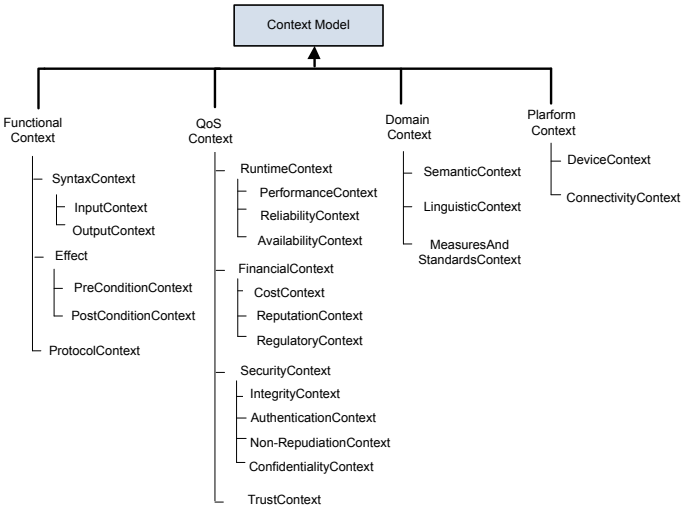

Fig. 3 Context model – taxonomy

– Protocol: a protocol is a consistent exchange of messages among services involved in dynamic service composition in achieving goals. The protocol context includes conversational rules which detail protocols of service invocations to achieve goals and context on data flows.

Quality of service (QoS) context describes non-functionality aspects determining the delivered quality of a service.
Runtime context attributes relate to the measurement of properties of the execution of a service.

– Performance: measurement of the time behaviour of services in terms of response time, throughput, etc.

– Reliability: ability of a service to be executed within the expected time frame.

– Availability: probability that the service is accessible.

Financial or Business context attributes allow the assessment of a service from a financial or business perspective.

– Cost: the amount of money required for provision and execution.

– Reputation: measures the service provider's trustworthiness.

– Regulatory compliance: a measure of how well a service is aligned with government or organizational regulations and policies.

Security context attributes describe service compliancy with security requirements.

– Integrity: protecting information from being deleted or altered in any way without the permission of the owner of that information.

– Authentication: ensures that both consumer and provider identity is verified.

– Non-repudiation: the ability of the receiver to prove to a third party that the sender really did send a message.

– Confidentiality: protecting information from being read or copied by anyone who has not been explicitly authorized by the owner of that information.

Trust refers to the establishment of trust relationships between client and provider – a combination of technical assertions (measurable and verifiable quality) and relationship based factors (reputation, history of cooperation).

Domain context refers to domain-specific requirements for service interaction.

– Semantic: refers to semantic frameworks (i.e. concepts and their properties) in terms of vocabularies, taxonomies or ontologies.

– Linguistic: the language used to express queries, functionality, and responses.

– Measures: refers to local standards for measurements, currencies, etc.

Platform context captures the technical environment a service is executed in.

– Device: the hardware platform on which the service is provided.

– Connectivity: the network infrastructure used by the service to communicate.

3.4 Enhanced context model aspects

We used a taxonomy (a hierarchy) to align context categories in the core context model. However, there are many types of non-taxonomic relationships between context categories that form a richer model. One context category can depend on different context categories in different cases, thus creating non-taxonomic relationships. Taxonomic relationships are defined in subsumption relationships. Non-taxonomic relations are mostly aspect-specific, i.e. local or non-local in terms of the hierarchy of the context model. Here are some examples to illustrate local and non-local relationships, which complement the taxonomic relations:

- Local : The *SecurityContext* is the integration of Integrity, Authentication, Non-repudiation and Confidentiality contexts. Different levels of each factor can bring different levels of security. All the constituent context categories are local to SecurityContext.
- Non-local : The *TrustContext* is a combination of technical assertions (measurable and verifiable quality) and relationship-based factors (reputation, history of cooperation). The constituent context categories of TrustContext are distributed in the context taxonomy and not local in the TrustContext. We can observe that some of the non-local relationships have dependencies. For example, the TrustContext has relationships with measurable and verifiable aspects.

Trust can be defined in various ways in different cases [26]. For example, a requester and provider interact through an exchange of encrypted and signed messages accompanied by additional trust information to establish identity and trust context of each participant. A Web service, which is guaranteed as a secure and reputable service from a reputable organisation can be considered as a trusted service. Another QoS aspect that can be defined through non-taxonomic relationships is software dependability, often defined as a combination of reliability and availability aspects [1], but also sometimes a variety of other criteria decided by software architects [53]. This adds to our point that not all context aspects can and should be fixed in one context model. We have also illustrated the links between location, language and other domain context aspects on the one hand and functionality aspects on the other hand in the banking localisation example earlier.

We cannot to define all cases formally. Instead, software architects can use the proposed techniques in Sections 4 and 5 to develop their specific non-taxonomic definitions. Our context model provides an abstract terminological framework, which needs to be customised in concrete situations. To illustrate this, in the earlier stock market example, we can identify a number of concerns that would require consumer and provider to negotiate their context needs and provisionings. For the Domain category, relevant aspects are:

- semantics: Standards were referred to (which act as simple, shared ontologies) such as GS1 or EANCOM
- lingual: English and German were used as languages (EN, DE)
- units: Currencies were used such as Euro and Dollar
- business: Reputation could have been considered

For the Platform category, relevant aspects are:

- platform: Mobile versus fixed access could have been considered
- connection: Wireless and secured could be a setting connecting platform and security aspects

While fully independent aspects do not cause problems, these non-taxonomic dependencies need attention in terms of modelling:

- within the QoS category: trust can be defined as a mix of reputation and security; dependability as a combination of reliability and availability.
- across categories, e.g. between Domain and Functionality, we find non-trivial dependencies that link function, semantics and linguistics in the form of standards-compliant interfaces (e.g. GS1).

For the latter, a rule could automatically derive settings depending on location. The location determines prices, which occur as units in the domain, but also as data in the functionality context. These non-taxonomic dependencies would require to translate service data between languages, e.g. from English into German - "Quote" to "Angebot" - based on standards like EANCOM or document-related attributes based on the GS1 standard for documents[3]. We could transform data between standards or their variants, e.g. "Quote" translates to "FullQuotation" based on a transformation between different EDIFACT variants and subsets such as EANCOM, EDIKEY, or EDIFICE. Other examples are transformations of currencies, e.g. conversion from Ireland (Eurozone) into UK (Pound Sterling) or transformation of rules and procedures, e.g. access rights to enable regulatory compliance by enabling legally required recording of activities through service adaptation.

4 Context Representation and modelling

We formalise the dynamic service context model as an ontology to support context representation and reasoning. An ontology consists of entities, relations, functions, axioms and instances. Context categories in the context model have taxonomic and non-taxonomic relations that can be formalised in a context model ontology. We start this section by detailing reasons to select OWL and its underlying foundations and introduce ontology-based modelling in Section 4.1. Based on this, we formalise the context model and specify some properties formally in description logics and OWL in Sections 4.2 and 4.3.

[3] For illustration, we can use the EDIFACT (United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport, http://www.unece.org/trade/untdid/welcome.html) and GS1 standards (supply and demand chains globally and across multiple sectors, http://www.gs1.org/).

4.1 Ontologies and ontology-based modelling

In existing context-aware systems, notations like XML, XM-based CC/PP [19], UML [31], Topic Maps [24], RDF [37], and OWL [56] are used for context modelling. We use the Ontology Web Language (OWL) to formalize context relationships based on the underlying description logic (DL) representation. The context model ontology further supports context reasoning, which is not adequately developed in the Web services domain [52]. The choice of OWL is motivated by its reasoning support. It provides language support for reasoning (OWL-DL) and supports SWRL (Semantic Web Rule Language) to enable rule-based reasoning. The logical language (DL) supports context composition and context constraints enhancements. OWL facilitates the sharing of conceptualisations (here the context between consumers and providers), which is important for cross-organizational service compositions.

The core elements of the description logic used as an underlying abstract language shall be introduced. The Attributive Language with Complements ($\mathcal{ALC}$) is the basis of many description logic languages. The OWL-DL, the description logic variant of OWL corresponds to $\mathcal{SHOIN}(\mathcal{D})$ [29], a description logic language based on $\mathcal{ALC}$ with transitive roles, role hierarchies, nominals (enumerated classes of object value restrictions), inverse properties, cardinality restrictions and concrete data types. In order to encode context aspects in $\mathcal{SHOIN}(\mathcal{D})$, and eventually in OWL-DL, an introduction of the constructors for $\mathcal{SHOIN}(\mathcal{D})$ is necessary. The constructors are illustrated in Table 1 [22]. Their semantics is based on the usual interpretations of first-order logic. C denotes concepts and R denotes property relationships.

Table 1 $\mathcal{SHOIN}(\mathcal{D})$ notation for the context ontology

Constructor	$\mathcal{SHOIN}(\mathcal{D})$	OWL-DL
conjunction	C1⊓C2	intersectionOf(C1,C2)
disjunction	C1⊔C2	unionOf(C1,C2)
negation	¬C1	complementOf(C1)
exists restriction	∃R.C	someValuesFrom(C)on(R)
value restriction	∀R.C	allValuesFrom(C)on(R)
atleast restriction	≥nR	minCardinality(n)on(R)
atmost restriction	≤nR	maxCardinality(n)on(R)

A DL specification can be constructed as a set of axioms. The basic constructors of $\mathcal{SHOIN}(\mathcal{D})$ can be used with either the subsumption $\sqsubseteq$ or equivalence $\equiv$ symbols to create DL statements. Axioms can be terminological axioms (TBox) or assertional axioms (ABox). Terminological axioms (statements about entities such as concepts and roles, but not individuals) can be subsumption or equivalence axioms. Assertional axioms (pertain only to individuals) can be concept assertions or role assertions axioms. A *subsumption axiom* gives necessary conditions for some a concept to be included (subclassed) in another, e.g. $A \sqsubseteq B$ where A, B are concepts. An *equivalence axiom* has the form $A \equiv B$. A *concept assertion* is of the form $C(i)$ where C is a concept from a TBox and i is an individual. A *role assertion* is of the form $R(a, b)$, where R is some role from a TBox and a and b are individuals.

4.2 Ontology-based context modelling

Our context model ontology consist of concepts (called classes in OWL terminology), their properties in the form of roles and individuals. The DL constructors and axioms can be used to formalize the context model ontology. These logical relations support composition and reasoning aspects. *Concepts* (OWL classes) of the context model are interpreted as sets of descriptive individuals. *Roles* (OWL properties) are binary relations on individuals. *Individuals* represent context (concept or role) instances.

Subsumption expresses whether a contextual concept/role is a subconcept/role of another concept/role. Subconcepts specialise (are subsumed by) their superconcepts. It uses context instance subsumption based on the hierarchical relationships of context in the context model. Classes can be organised into a concept hierarchy that formalises the context model taxonomy described earlier on. For example, the Input Parameter context is a subconcept of the Functional context. This means all members of the Input Parameter context are members of (subsumed by) the concept Functional context. Subsumption can be used to match consumer context requirements against provider context (later called service profiles) and to determine configurables (service selection and process composition), i.e. comparing user requirements against actual or declared provider properties (through satisfaction and matching). Constraints compare actual and required context properties. Both structural (subconcept) and logical (implication) subsumption relationships can be determined automatically.

Now we look into concept and role formalisation in detail to specify further characteristics of the context model beyond taxonomical subsumption relationships.

4.2.1 Concept description

The building blocks of an OWL ontology are classes that represent concepts. $\mathcal{SHOIN}(\mathcal{D})$ axioms can be

used to specify complex class descriptions – classes could be a subclass or disjoint with other classes:

- **Subclasses** represents hierarchical relationships between classes (subsumption). For example, Integrity is part of Security, i.e. $Security \sqsupseteq Integrity$.
- **Disjointness** means that individual components are different. For example, high security and performance is hard to achieve, i.e. $Security \sqcap Performance \equiv \bot$ (an oversimplification to illustrate the concept).
- **Completeness** means that a context is built from only pre-specified contexts. For example, security is an integration of four aspects: $Security \equiv Integrity \sqcap Authentication \sqcap Non\text{-}repudiation \sqcap Confidentiality$.
- **Composed class descriptions:** The composition of more than one context category can be described in complex class descriptions, e.g. the effect context can have either a pre-condition context or a post-condition context or both, i.e. $Effect \sqsupseteq Pre\text{-}Condition \sqcup Post\text{-}Condition$. The platform context may have to have both device context and connectivity context, i.e. $Platform \equiv Device \sqcap Connectivity$. These can be used to further restrict subsumption relationships between *Effect* and *Platform* and their subclasses.

4.2.2 Role description

Context in the taxonomy can have properties, which can be formalized within the context model ontology. Roles represent relationships between individuals or an individual and data literals. Here, individuals are context instances. Generally, a role can be an object role, datatype role or annotation role based on how they are used within the ontology. *Object roles* link an individual to an individual, e.g. *S.Security hasPart S.Integrity* for service *S*. *Datatype roles* link an individual to an XML schema datatype value or an RDF literal, e.g. *D.Device hasDisplaySettings "6x8"* for device *D*. *Annotation roles* add meta-information to contextual concepts, individuals and object/datatype roles.

A role can also be categorised in terms of its function. *Generic* roles that are hard-coded into the context model are *hasPart* or *hasLevel*. Some roles are *aspect-specific* – *hasDisplaySettings* is an example for the device aspect. The second category is introduced to further qualify context. This could have been done as sub-concept roles with typed instances, but here they are part of the vocabulary.

4.2.3 Rule-based context derivation

Derived context is implicit context derived from explicit context in the context ontology based on rules in the form $Antecedent \rightarrow Consequent$. Antecedent and consequent consist of one or more context concepts and role descriptions. For example, if in a client context a mobile device is indicated in the respective context aspect, the output message display should be matched with the display settings of the device:

$$hasMessage\ (client,message) \wedge hasDevice\ (client,mobile) \rightarrow hasDisplaySetting\ (message,3x5).$$

These rules can be implemented as SWRL rules [30].

A derived context can affect other context aspects. For instance, deriving an implicit Security context based on a given explicit context of Integrity and Confidentiality is illustrated below. The rule

$$Service(?s) \wedge objectPropertyHasIntegrity(?s,?x) \wedge objectPropertyHasConfidentiality(?s,?x) \wedge swrlb{:}stringEqualIgnoreCase(?x,"high") \rightarrow objectPropertyHasSecurity(?s,?x)$$

means that if a service provides integrity and confidentiality, then it is considered secure. So, for the explicit context of a service

$$<objectPropertyHasIntegrity\ rdf{:}resource="high"/>$$
$$<objectPropertyHasConfidentiality\ rdf{:}resource="high"/>$$

we get as derived output

$$<objectPropertyHasSecurity\ rdf{:}resource="high"/>$$

In order to achieve security, extra processing time for a service might be needed, specified by the rule: if security is high then response time is greater than 100 ms.

$$Service(?s) \wedge objectPropertyHasSecurity(?s,?x) \wedge swrlb{:}stringEqual(?x,\ "high") \wedge dataTypePropertyHasResponseTime(?s,?y) \rightarrow swrlb:GreaterThan(?y,\ 100ms).$$

4.3 OWL-based context formalisation

An excerpt from the context model ontology in OWL-DL is illustrated below. Lines 1-8: Performance is a subclass of the Runtime context and Runtime context is a subclass of the Quality context. Lines 9-13: *hasResponseTime* is a (functional) data type property. Lines 15-19: the Cost context is a subclass of the attributes defining the Financial context and the latter is a subclass of the Quality context. Lines 20-24: *hasCostValue* is a functional data type property. Lines 25-36: the Security context is a subclass of Quality; Integrity and Confidentiality are subclasses of Security. Lines 37-52: Security has an object property *hasPart* and inverse *isPartOf* based on the *unionOf* (Integrity, Confidentiality).

```
1 <owl:Class rdf:ID="PerformanceContext">
2     <rdfs:subClassOf rdf:resource="#RuntimeContext"/>
3 </owl:Class>
4 <owl:Class rdf:ID="RuntimeContext">
5     <rdfs:subClassOf>
6          <owl:Class rdf:ID="QualityOfServiceContext"/>
7     </rdfs:subClassOf>
8 </owl:Class>

9 <owl:DatatypeProperty rdf:ID="datatypeProperty_hasResponseTime">
10     <rdfs:range rdf:resource="http://.../XMLSchema#string"/>
11     <rdfs:domain rdf:resource="#PerformanceContext"/>
12     <rdf:type rdf:resource="http://../owl#FunctionalProperty"/>
13 </owl:DatatypeProperty>

14 <owl:Class rdf:ID="CostContext">
15     <rdfs:subClassOf rdf:resource="#FinancialContext"/>
16 </owl:Class>
17 <owl:Class rdf:ID="FinancialContext">
18     <rdfs:subClassOf rdf:resource="#QualityOfServiceContext"/>
19 </owl:Class>

20 <owl:DatatypeProperty rdf:ID="datatypeProperty_hasCostValue">
21     <rdfs:domain rdf:resource="#CostContext"/>
22     <rdf:type rdf:resource="http://../owl#FunctionalProperty"/>
23     <rdfs:range rdf:resource="http://../XMLSchema#string"/>
24 </owl:DatatypeProperty>

25 <owl:Class rdf:about="#SecurityContext">
26     <rdfs:subClassOf rdf:resource="#QualityOfServiceContext"/>
27 </owl:Class>
28 <owl:Class rdf:ID="Integrity">
29     <rdfs:subClassOf>
30          <owl:Class rdf:about="#SecurityContext"/>
31     </rdfs:subClassOf>
32 </owl:Class>
33 <owl:Class rdf:ID="Confidentiality">
34     <rdfs:subClassOf>
35          <owl:Class rdf:about="#SecurityContext"/>
36     </rdfs:subClassOf>

37 <owl:ObjectProperty rdf:ID="objectProperty_hasPart">
38   <owl:inverseOf>
39    <owl:ObjectProperty rdf:ID="inv_of_objectProperty_isPartOf"/>
40   </owl:inverseOf>
41   <rdfs:domain rdf:resource="#SecurityContext"/>
42   <rdf:type rdf:resource="http://.../owl#FunctionalProperty"/>
43   <rdfs:range>
44        <owl:Class>
45             <owl:unionOf rdf:parseType="Collection">
46                  <owl:Class rdf:about="Confidentiality"/>
47                  <owl:Class rdf:about="Integrity"/>
48             </owl:unionOf>
49        </owl:Class>
50   </rdfs:range>
51 </owl:ObjectProperty>
```

The OWL representation of cost, response time, lingual and security context features of a sample service is illustrated below. These properties are model instances, i.e., individual context constraints that can be monitored and validated at runtime. Lines 3-5: the object property *hasSecurity* of Service 1 denotes security attributes, here abstracted as Security-S1. Lines 7-12: the object property *hasMaxResponseTime* of Service 1 required to be less than 3 milliseconds. Lines 14-19: the property *hasCost* of Service 1 requests 0.1 Euro. Lines 21-26: the output of Service 1 has a data type property, *hasLanguage*, referring to English, which could reflect the US-locale from the use case scenario in Section 2.

```
1<Service rdf:ID="Service-S1">
2
3  <objectProperty_hasSecurity>
4      <SecurityContext rdf:ID="Security-S1"/>
5  </objectProperty_hasSecurity>
6
7  <objectProperty_hasMaxResponseTime>
8      <Performance rdf:ID="Performance-S1">
9         <datatypeProperty_hasResponseTime
          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
10        < 3 ms </datatypeProperty_hasResponseTime>
11     </Performance>
12 </objectProperty_hasMaxResponseTime>
13
14 <objectProperty_hasCost>
15    <Cost rdf:ID="Cost-S1">
16       <datatypeProperty_hasCostValue
         rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
17       0.1 Euro </datatypeProperty_hasCostValue>
18    </Cost>
19 </objectProperty_hasCost>
20
21 <objectProperty_hasOutput>
22     <Output rdf:ID="Output-S1">
23       <datatypeProperty_hasLanguage
         rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
24       English </datatypeProperty_hasLanguage>
25     </Output>
26 </objectProperty_hasOutput>
27
28</Service>
```

5 Context Manipulation

Often, a context specification needs to be adapted for further processing or several contexts, e.g., of different services in a process, need to be combined. We provide an operator calculus for context specifications to facilitate these manipulations [42]. While techniques for adaptation and match-making itself are not the focus of this investigation, the context model framework shall provide a foundation for these.

5.1 Context model specification and service context profiles

Before addressing the manipulation of context, the notion of a context specification and its semantics need to be made precise. We assume the **context model** to be a DL specification based on the $\mathcal{SHOIN}(\mathcal{D})$ subset from Table 1, $Context\ Model = \langle \Sigma, \Phi \rangle$. For instance, for the *SecurityContext*, we define Σ and Φ as follows.

$$\Sigma = \langle \{IntegrityContext, AuthenticationContext, ...\}; \{hasPart, isPartOf\} \rangle$$

$$\phi = \left\{ IntegrityContext \overset{isPartOf}{\rightarrow} SecurityContext \right\}$$

We assume in general the following signature inclusion $T \subset \Sigma$ for all signatures Σ where T is the *context model taxonomy* signature, as defined in Section 3.3, where: $\Sigma = \langle \{FunctionalContext, QoSContext, ...\}; \{hasPart, isPartOf, hasCost, hasSecurity, ...\} \rangle$ and the taxonomy $T = \langle \{FunctionalContext, QoSContext, ...\};$

$\{hasPart, isPartOf, ...\}\rangle$. If the taxonomy is not adhered to or other changes or extensions take place, context modelling might require syntactical elements to be renamed (we will provide a respective operator later).

The $Context\ Model = \langle \Sigma, \Phi \rangle$ can be interpreted by a set of (algebraic) models M. The model notion [32] refers to algebraic structures that satisfy all context descriptions ϕ in Φ. The set M contains algebraic structures $m \in M$ with

- instances C^I for each contextual concept (class) C,
- roles $R^I \subseteq C_i^I \times C_j^I$ for all context roles $R : C_i \rightarrow C_j$

such that m satisfies the context description. We define the satisfaction relation over the selected connectors of the description logic $\mathcal{SHOIN}(\mathcal{D})$ from Table 1. The assumptions and limitations that apply to algebraic specifications in general (e.g. decidability) apply.

A **context specification** is application-specific and has instances and instance-level axioms, i.e. *Context Specification* $\in$ *Context Model*, where *Context Model* $= \langle \Sigma, \Phi \rangle$. The **consistency** of a **context specification** ensures that a context model does not contain any contradictory facts. A context specification is **consistent**, if there are models that satisfy the specification. Based on the descriptions of a contextual concept, a reasoner can check whether or not it is possible for a concept to have any instances. A concept is deemed to be inconsistent if it cannot have any instances.

We use the notion of a **service context profile** (SCP) to extend the context specification notion for a Web service and denote its incarnation at runtime. A service context profile SCP captures context model instances of individual services, i.e. adding instance-level axioms to the context specification. The context model provides a contextualization framework, in which service-related context aspects are captured. An SCP is represented as an association of values (instances) to context model aspects

$$\begin{aligned} SCP = [&\{F(1)...F(n_F)\}, \\ &\{Q(1)...Q(n_Q)\}, \\ &\{D(1)...D(n_D)\}, \\ &\{P(1)...P(n_P)\}] \end{aligned}$$

where $\{F(1)...F(n_F)\}$ are functional context instances, $\{Q(1)...Q(n_Q)\}$ are quality of service context instances, $\{D(1)...D(n_D)\}$ are domain-based context instances, and $\{P(1)...P(n_P)\}$ are platform-based context instances. Each of the instance elements is typed by the respective context model aspect. A code excerpt of a service context profile was illustrated earlier on in Section 4.3.

5.2 Context manipulation operators

We introduce *context manipulation operators*, before addressing *context composition operators* in Section 5.3. The latter can be distinguished from the normal manipulation operators as they preserve the internal composition structure (i.e. are reversible). The consistency of context specifications is a concern. We will point out and (informally) prove key properties with respect to consistency preservation.

We have two types of *context manipulation operators – service-level* and *process-level*. At service-level, we discuss context aspects relevant to individual services, e.g., manipulating different context aspects of single service. At process-level, we discuss context aspects relevant to contextualized service processes, e.g., manipulating a single context aspect relevant to different services in a process. We define three fundamental context manipulation operators for service-level context manipulation. They are **Renaming**, **Restriction**, and **Refinement**. We also define two operators, **Union** and **Intersection** for process-level context manipulation. We discuss the consistency preservation of context specifications by the operators. We use DL-level formalisms to define context manipulation operators.

5.2.1 Service-level context manipulation

Renaming. If the taxonomy is not adhered to or other changes or extensions take place, context modelling might require syntactical elements to be renamed. A **Renaming** operator can be defined element-wise for a given signature Σ. By providing mappings for the elements that need to be modified, a new signature Σ' is defined,

$$\Sigma' \stackrel{\text{def}}{=} \Sigma\,[n_1 \mapsto n_1',\ \ldots\ , n_m \mapsto n_m']$$

for all concepts or roles $n_i (i = 1, \ldots, m)$ of Σ that need to be modified. For example, concepts *OSContext* is used instead of *PlatformContext* and roles *hasOperatingSystem* is used instead of *hasPlatform*.

$$\begin{aligned} \Sigma' = \Sigma\ [\ &\{PlatformContext \mapsto OSContext\}; \\ &\{hasPlatform \mapsto hasOperatingSystem\}\ ] \end{aligned}$$

Restriction. While context specifications are often used 'as is', it is sometimes desirable to focus on specific parts. Restriction is an operator that allows context combinations to be customised and undesired elements (and their roles) to be removed. A restriction can be expressed using the **Restriction** operator $\langle \Sigma, \Phi \rangle_{|\Sigma'}$ for a context specification, defined by

$$\begin{aligned} \langle \Sigma, \Phi \rangle_{|\Sigma'} \stackrel{\text{def}}{=} \langle \Sigma \cap \Sigma', \{\phi \in \Phi \mid\ & rls(\phi) \in rls(\Sigma \cap \Sigma') \wedge \\ & cpts(\phi) \in cpts(\Sigma \cap \Sigma')\}\rangle \end{aligned}$$

with the usual definition of role and concept projections $rls(\Sigma) = R$ and $cpts(\Sigma) = C$ on a signature $\Sigma = \langle C, R\rangle$. For example, if an integrity context of a service is a concern instead of the complete security context, then this can be specified as

$$rls(\phi) = \{hasIntegrity\} \text{ and}$$
$$cpts(\phi) = \{IntegrityContext\}$$

Consistency preservation is an important property. Restriction *preserves consistency*, which holds as constraints are, if necessary, removed. Restriction can be applied in combination with any context combinator such as Intersection, Union or Refinement.

Refinement. Consistency is a requirement that should apply to all combinations of ontologies. A typical situation is the derivation of a new context from an existing one [6]. We introduce a constructive operator, **Refinement**, which is a consistent (i.e., consistency-preserving) extension in terms of contextual concepts and roles. The Refinement can be linked to the subsumption relation and semantically constrained by an inclusion of interpretations (models that interpret a context). Refinement preserves existing roles, e.g., the satisfiability of the original context specification. As the original contextual concept and role types cannot be further constrained, the extension is consistent.

The consistency-preserving Refinement operator provides a constructive subsumption variant that allows

- new subconcepts and new subroles to be added, and
- new constraints to be added, if these apply consistently to the new elements.

Assume a context specification $C = \langle \Sigma, \Phi\rangle$. For any specification $\langle \Sigma', \Phi'\rangle$ with $\Sigma \cap \Sigma' = \emptyset$, we define a **Refinement** of C by $\langle \Sigma', \Phi'\rangle$ through

$$C \oplus \langle \Sigma', \Phi'\rangle \stackrel{\text{def}}{=} \langle \Sigma + \Sigma', \Phi + \Phi'\rangle$$

We can *demonstrate consistency preservation.* The pre-condition $\Sigma \cap \Sigma' = \emptyset$ implies $\Phi \sqcap \Phi' = \bot$, i.e. consistency is preserved, which is an important property for dynamic, automated environments. In this situation, existing roles of $C = \langle \Sigma, \Phi\rangle$ are inherited by $C \oplus \langle \Sigma', \Phi'\rangle$. Existing roles can be refined as long as consistency is maintained, which might require manual proof in specific situations that go beyond the operator-based application.

Refinement can be used to adapt provider context to a context signature Σ' and a context description Φ', e.g. to add device aspects to a context $\langle \Sigma', \Phi'\rangle$ if the user's device context supports a given feature:

$$\langle \Sigma', \Phi'\rangle \oplus \langle \{DeviceContext, FeatureContext\}, \{hasDevice, hasFeature\}\rangle$$

5.2.2 Process-level context manipulation

Adding a context specification to another specification or removing specific context roles from a context specification is often required, particularly if service contexts are combined within a process. The operators **Union** and **Intersection** deal with these situations, respectively. Two context specification $C_1 = \langle \Sigma_1, \Phi_1\rangle$ and $C_2 = \langle \Sigma_2, \Phi_2\rangle$ can be considered (generally associated to two different services) in a process.

- The **Intersection** of C_1 and C_2, expressed by $C_1 \cap C_2$, is defined by

  $$C_1 \cap C_2 \stackrel{\text{def}}{=} \langle \Sigma_1 \cap \Sigma_2, (\Phi_1 \cup^+ \Phi_2)|_{\Sigma_1 \cap \Sigma_2}\rangle$$

  We describe the $\cup^+$ operator for context specification later in this section, which is defined on a case by case basis for different context aspects. Intersection is semantically defined based on an intersection of context interpretations, achieved through projection onto common signature elements.
- The **Union** of C_1 and C_2, expressed by $C_1 \cup C_2$, is defined by

  $$C_1 \cup C_2 \stackrel{\text{def}}{=} \langle \Sigma_1 \cup \Sigma_2, (\Phi_1 \cup^+ \Phi_2)|_{\Sigma_1 \cup \Sigma_2}\rangle$$

  Union is semantically defined based on a union of context interpretations.

Note, this assumes sequential process composition. The operators could also be integrated with common semantics for conditional or iterative control flow constructs.

Again, consistency is a crucial property and we need to *demonstrate consistency preservation.* Both Union and Intersection operations can result in consistency conflicts, but that the combination of two context specifications of two services is conflict-free, i.e. semantically, no contradictions should occur, can be shown as follows. A consistency condition can be verified by ensuring that the set-theoretic interpretations of two contexts C_1 and C_2 are not disjoint, $C_1^I \cap C_2^I \neq \emptyset$, i.e. their combination is satisfiable and no contradictions occur.

The **combination operator** $\cup^+$ deals with process-level composition of context aspects in terms of the types of context aspects involved. The combination mechanism, which is the functionality of the $\cup^+$ operator, differs between context aspects. We address all context aspects in our context model ontology to define a complete list of $\cup^+$ operators. $C(i)$ refers to the context aspect value of service i, e.g., $C(i)$ for the service i can equal to 600(ms) for the response time aspect.

- The **Lowest Common Denominator** $(\cup^+_{LCD})$ $\cup^+_{LCD} \rightarrow Min_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ.
  Example: for a security aspect, the overall security of a process is determined by the weakest security setting of all individual services.

- The **Least Common Subsumer** [15] $(\cup^{+}_{LCS})$
  $\cup^{+}_{LCS} \rightarrow \bigcap^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: for the language aspect, the least common subsumer of all individually used languages are the language(s) common to all (intersection).
- The **Logical OR** $(\cup^{+}_{OR})$
  $\cup^{+}_{OR} \rightarrow \texttt{OR}^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: for the deployment environment, the service deployment environment needs secure internet connection or connection bandwidth greater than 10Mbps.
- The **Accumulation** $(\cup^{+}_{ACC})$
  $\cup^{+}_{ACC} \rightarrow \Sigma^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: The cost of a process is an accumulation through summation of the cost of each service.
- The **Logical AND** $(\cup^{+}_{AND})$
  $\cup^{+}_{AND} \rightarrow \texttt{AND}^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: for the deployment environment, the service deployment environment needs Windows operating system and connection bandwidth greater than k Mbps.
- The **Mediation** $(\cup^{+}_{MED})$
  $\cup^{+}_{MED} \rightarrow \texttt{MED}^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: in service composition, if an output context (boolean: true or false) of a service S_j is composed with an input context (integer: 0 or 1) of a service S_{j+1}, then a mediation is needed. Mediations are represented as mappings.

In order to illustrate this for a service process P, we assume P has two services S_i and S_j and corresponding context specifications SCP_i and SCP_j. Both specifications are characterised in terms of five context aspects (in-parameter, out-parameter, response time, security and language, respectively).

$$SCP_i = [int, bool, 1ms, 1111, EN] \text{ and}$$
$$SCP_j = [int \times int, int, 10ms, 1001, FR]$$

The aim is to combine SCPs to process-level contexts using the different $\cup^{+}$ variants:

- *in, out* – sequential composition, which is a causal structural composition (mediation). Correctness of this composition is a concern. We address this type of composition further in Section 5.3.
- *cost, performance* – numerical composition through addition (accumulative).
- *security* – the lowest common denominator, which is a kind of intersection for security settings.
- *language* – intersection as the composition principle.

The results of the combination can be illustrated as follows. Assume a service process P:

$$P = \{S_i, S_j\} \text{ with } \langle \Sigma, \Phi \rangle_p = \langle \Sigma, \Phi \rangle_{S_i} + \langle \Sigma, \Phi \rangle_{S_j}$$

i.e.

[int,bool,1ms,1111,EN] + [int×int,int,10ms,1001,FR]

The composition can be illustrated as:

$$[\text{bool} \cup^{+}_{MED} \text{int} \times \text{int}], [1\text{ms} \cup^{+}_{ACC} 10\text{ms}],$$
$$[1111 \cup^{+}_{LCD} 1001], [\text{EN} \cup^{+}_{LCS} \text{FR}]$$

The results of the individual aspect combinations are:

$$[1\text{ms} \cup^{+}_{ACC} 10\text{ms}] = 11 \text{ ms}, [1111 \cup^{+}_{LCD} 1001] = 1001$$

5.3 Context composition

The explicit support for context composition is important for service context profiles in composed service processes. As an extension to the context manipulation operators, we introduce two types of composite operators for context specifications. In contrast to Union and Intersection, context composition retains subcomponents as identifiable parts of the result and, therefore, makes composition reversible.

The *subsumption* is the central relationship in ontology languages, allowing context taxonomies to be defined in terms of subtype relationships [2]. The *composition* is a fundamental relationship that describes the part-whole relationship between concepts or instances (individuals) [44]. Composition is less often used in ontological modelling languages. The notion of composition shall be applied for context in two different ways:

- *Structural (service-level) composition.* The structural hierarchies define an important aspect of context [17]. The structural composition can be applied for instance for input/output or for security with its subaspects confidentiality or availability. In the latter case, composition is more adequate than seeing these as subtypes if their later implementation through different system components is considered.
- *Sequential (process-level) composition.* Dynamic elements (services) can be composed to represent sequential process behaviour. While context does not directly represent behaviour, service context models need to be aggregated along with the behavioural composition of services in a process.

We use the symbol "$\rhd$" to express composition. The composition is syntactically used in the same way as subsumption "$\sqsubseteq$" to relate context descriptions.

- *Context composition hierarchies* can consist of unordered subcomponents, expressed using the component composition operator "$\rhd$". An example is *Security* $\rhd$ *Confidentiality*, meaning that a *Security* aspect consists of *Confidentiality* as a part. The components can be interpreted by unordered multi-sets.

The structural composition $C \rhd \{D_1, \ldots, D_n\}$ is defined by $C \rhd \{D_1\} \sqcap \ldots \sqcap C \rhd \{D_n\}$ where $C \rhd \{D\}$ means that C is structurally composed of D if C and D are context specifications. The parts $D_i, i = (1, .., n)$ are not assumed to be ordered. The structurally composed concepts are interpreted as multi-sets.

– *Service processes* can be *sequences* that consist of ordered process elements, again expressed using the composition operator "$\rhd$". An example is *Process* $\rhd$ *Service*, meaning that *Process* is actually a composite service, which contains for instance a *Service* element. We see composite process implementations as being interpreted as ordered tuples providing a notion of sequence. More complex behavioural compositions are not covered here. The sequential composition $C \rhd [D_1, \ldots, D_n]$ is defined by $C \rhd [D_1] \sqcap \ldots \sqcap C \rhd [D_n]$ where $C \rhd [D]$ means that C is sequentially composed of D if C and D are services. The sequentially composed concepts are interpreted as tuples. The parts D_i with $i = (1, .., n)$ are assumed to be ordered with $D_1 \leq \ldots \leq D_i \leq \ldots \leq D_n$ describing an execution ordering $\leq$ on the D_i.

Note, that the composition operators are specific to the respective element types, whereas subsumption is generic.

While the subsumption relationship is defined through subset inclusion, the composition relationships are defined through membership in collections (multi-sets for structural composition and tuples for behavioural composition).

6 Application and Discussion

While the earlier stock market use case served to introduce the context notion for a single service, we now use a more process-oriented case study to illustrate the *application of the context model* and the supporting *calculus* for service-based process compositions using simple context constraints here.

We also discuss validation and the runtime infrastructure to support dynamic service processes in Section 6.1.

6.1 Applicability in case study

This utility bill pay scenario assumes that Dublin-based user pays a utility bill from his UK bank account using an enterprise client. The enterprise client (e-client) satisfies user requests by combining heterogeneous services *Billing Service*, *Banking Service* and *Payment Confirmation Service* at process run-time. We assume that service providers charge enterprise clients (e-client) for provided services.

In this scenario, there are two simple constraints, which are dynamically generated based on service-level agreements the client has with the service providers. Firstly, the total cost of the process should be less than 0.5 Euro and, secondly, the process response time should be less than 2 seconds. The cost of each service can change based on exchange rates and the response time of each service can only be measured at run-time. We can also see a semantic mismatch of output and input parameters of *Billing Service* and *Banking Service*. Security validations, such as authentication may need to be done at process run-time. The context operationalisation of the Web service process is described in Figure 4.

We assume that each constituent service is attached to a service context profile (SCP) that characterizes a service with its context instance information. An SCP is an instance-level specification of the context model ontology, see Section 5.1. The cost of a service might be fixed when services are composed to a service process. However, the response time of a service can only be collected after the service execution at process runtime. We assume that the UK Banking Service and the Global Banking Service have the same interface as do the Email, Fax and Mobile Services. Services of the same interface can be assigned to a single dynamic partner link of a BPEL process at process runtime.

For the *Billing Service*, the user context constraint is verified as a precondition of the service, which is part of the functional context. The *Billing Service* outputs the utility bill in Euro. Then, the *Banking Service* is invoked and a user is asked to provide her/his banking details. This service has an encryption pre- and post-condition attached to it – both as part of the context of the *Banking Service*. After that, the following options need to be decided at runtime:

– In order to perform the banking transaction, the higher-level *Banking Service* invokes the subordinated *UK Banking Service*. If it fails, the *Global Banking Service* can replace it, but we assume that it has a higher response time and a lower cost than the original service. Its response time (defined by the provider as less than 200 ms) is considerably lower compared to the process response time, however the actual response time needs to be determined. The process response time needs to be validated. Small deviations of response time to the defined process response time constraint can also be acceptable, which needs to be addressed by the monitoring system.
– The *Pay Confirmation Service* is invoked after the *Banking Service*. Whether to invoke the *Email, Fax*

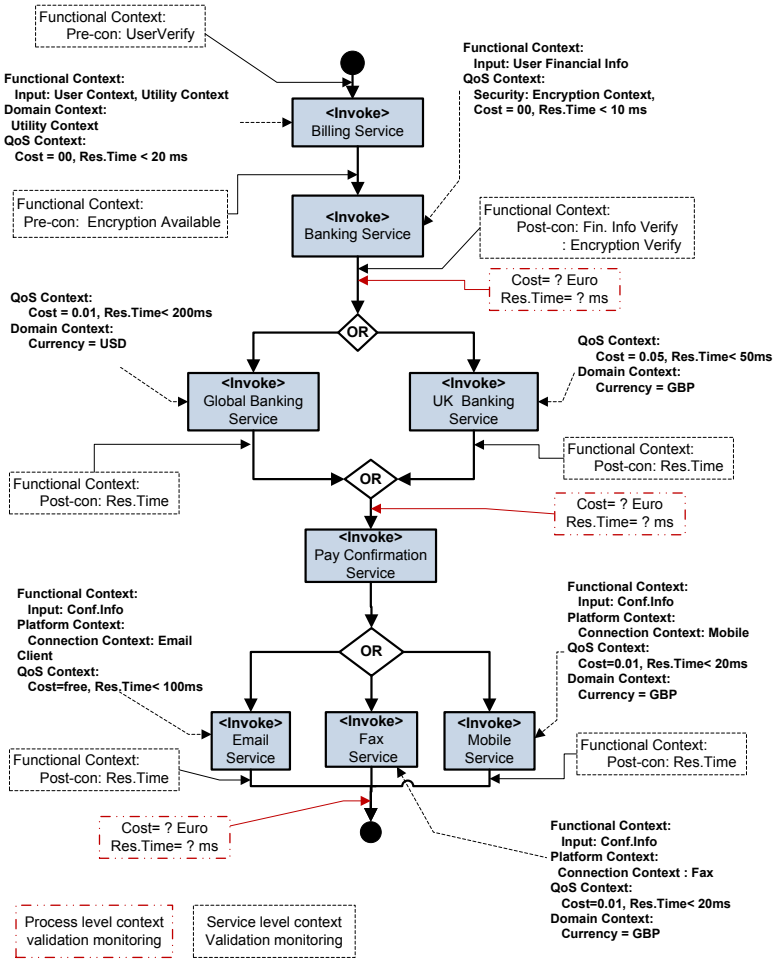

Fig. 4 Context operationalisation in a Web service process – SLA constraints

or *Mobile Service* is decided based on process constraints. If a *Banking* and *Fax Service* are deployed and the *Banking Service* fails, then a more costly banking service could be an option. In order to maintain process costs within the cost constraint, the process could replace the *Fax Service* with the *Email Service* at runtime (assuming email services are free).

If a service fails, then the cost constraint needs to be validated before a replacement, which is a pre-condition for that service. That is, the Web service process needs to be contextualized with the cost context to support pre-condition checks at runtime. These types of constraint validations work as pre-condition validations of the constituent services at runtime. The response time, on the other hand, could be estimated based on previous executions, but the exact response time context can only be measured at runtime and validated as a post-condition of constituent services. That is, Web service processes need to be operationalised with the response time context constraints to support runtime validation.

6.2 Validation and discussion

In the following, we will analyse the benefits of the proposed framework. Our context operationalisation scenario focuses on cost and response time context categories and validates the usefulness of the respective constructs. The aim is to validate the suitability of the framework constructs within the representative examples used and discuss how this could be utilised in a dynamic service monitoring and composition environment, specifically looking at context derivation and rule support. However, other context categories, such as security, device, location etc. are also utilized and have been illustrated before.

Subsumption has already been illustrated through examples regarding the security and parameter contexts in Section 4.2. *Subsumption reasoning* is important in provider and consumer context matching scenarios where the provider is required to be better, i.e. needs to subsume the requirements of the service user.

For instance, QoS values or functional types (in/out) need to be better. In practical terms, refinement is a useful constructive operator that implies subsumption.

Context derivation supports pre-condition validation of constituent services before invoking them at runtime. Suppose that a customer prefers mobile messages to emails and the service process deploys a mobile messaging service. Then, payment confirmation information needs to be adapted to the display settings of the device. The mobile messaging service may need the customer's mobile connection and device information such as TXT or MMS support. If payment confirmation information is sent as an MMS message, then the mobile connection as well as the device context needs to be derived and checked whether the connection supports MMS messaging. A *derivation rule* for device MMS settings can be defined as

$$\textit{(hasMessage(client, message)} \wedge \textit{ hasDevice(client, mobile)} \wedge \textit{ hasMMSSupport(mobile))} \rightarrow \textit{ hasDisplaySetting(mobile, MMSSetting).}$$

Pre-condition validation before invoking a Mobile Service at runtime would here utilize platform and device context. Context derivation can benefit from *role properties* such as symmetry or transitivity, e.g., the *hasPart* role is transitive or elements of a parameter can be identified as part of the interface and can be subjected to type constraints.

Composite constraints are needed for context validation. We consider two constraints on process cost and process response time, which are cumulative service-level context aspects defined using the *Union* operator. Adding the cost context and the response time context of each service is needed to find the process cost and process response time. For instance, the process response time context is defined as:

$$\textit{ResponseTime (Process)} = \textit{ResponseTime (Billing Service)} + \textit{ResponseTime (Banking Service)} + \textit{ResponseTime (UK Banking Service)} + \textit{ResponseTime (Pay Confirmation Service)} + \textit{ResponseTime (Mobile Service)}$$

Restriction and *refinement* are two other necessary context manipulation operators. Restriction is used to prepare a context specification (or a service context profile) for matching, e.g. to tailor provider context to focus on QoS aspects only,

$$\langle \Sigma, \Phi \rangle_{|\langle QoS, \emptyset \rangle}$$

for a context signature Σ and description Φ. Refinement can be used to adapt provider context, e.g. to add device aspects to the context specification $\langle \Sigma, \Phi \rangle$:

$$\langle \Sigma, \Phi \rangle \oplus \langle [\textit{Device,Platform}], [\textit{DisplaySettings}] \rangle$$

Structural composition allows us to distinguish a part-of hierarchy from a subsumption hierarchy. For instance, confidentiality is part of security, $\textit{Security} \rhd \{\textit{Confidentiality}\}$, which is an implementation perspective, where different security provisioning and monitoring concerns are attached at an implementation level. *Sequential composition* allows us to formalise the process illustrated in Fig. 4, which is a sequential composition of three services: $\textit{PaymentProcess} \rhd [\ \textit{BillingService, BankingService, PayConfirmationService}\ ]$.

Finally, we look at placing dynamic (context) aspects in a service process at runtime, which we have alluded to in the Introduction. We can generate validatable constraints as *context constraints*. Constraints are context-based restrictions. An *instrumentation* is based on weaving constraints and data collectors with a deployed service process. Service context profiles are the runtime representation (introduced at the end of Section 5.1 based on a respective OWL example in 4.3). Our work in [54], which allows context constraints to be woven into BPEL processes and checked dynamically, uses a policy constraints language PCPL (process customisation policy language). This allows dynamic context change to be detected (assuming respective probes being implemented). Different fault categories can be distinguished - cf. the boundary model advocated in [54]. If required, service (re-)composition can then take place.

7 Related Work

Context is used in various applications [16,34,27,23, 10], often to capture spatial and temporal aspects in mobile [50] and ubiquitous systems [28,33,46].

The *context notion* has been applied to define locative and temporal aspects in dynamic applications [28, 33,50]. Bronsted et al. [11] investigate composition approaches specifically for pervasive systems and single out the need for context-awareness. We already mentioned CONON [56] and SOUPA [25] as widely used context models for pervasive computing. Fundamental context classifications, such as device, location, person and activity for capturing information about the execution situation, are used. While these context models do not characterise dynamic aspects of services as software entities within processes, we have adopted taken on board these context aspects. A context notion and classification is also used to define functional and non-functional features of Web services [37,38] focusing primarily on design-time context matching for service selection. Rosemann et al. [47] investigate context in business processes in general and propose a conceptual context taxonomy, but acknowledge the need for further

research on process execution aspects, called the immediate context there.

The previous work on *context in pervasive and ubiquitous applications* uses context ontologies, which are tightly coupled with individual applications [21]. In their work, a context ontology is a part of application-dependent middleware. Our concern is a more general context-aware middleware support for dynamic service composition applications. The requirements attached to composition and execution of services at process run-time are the main concern. Our proposed context model is not tightly coupled with individual Web service applications and the context model facilitates a middleware support for dynamic service composition. Service providers can use a context model for developing context-aware services, which can also be organized in service communities proposed in [38,40].

While a context notion has been used widely for static environments, a context classification to address dynamic aspects of Web services such as *service composition* is still lacking [47], however, context for adaption is seen as having potential [35]. In previous work on service composition such as [47,37,49,51], context has been explored for service discovery and selection at design time, while we focus on context operationalisation at process runtime in order to validate dynamic requirements. A solution to context and context-aware Web services for Web service process domain is proposed in [37], which is about context-based service selection for service composition. Their context categorization is detailed and only lacks the domain aspects and interdependency support provided here, but it is primarily aimed at static context and does not address dynamic requirements-based aspects such as runtime properties. We reused their service properties as the starting point of our context model development and add dynamic context aspects. While our context model coincides in key aspects with theirs, their policy-based implementation framework does not instrument service processes with dynamic requirements, i.e. does not allow context policies to be validated dynamically. It also does not provide a rich modelling framework in terms of the operators and reasoning we presented. In [38], service clusters are described in a detailed classification. They detail static semantics, dynamic semantics, and also quality of operations. However, their focus is on semantic clusters for services that reinforces the concept of a service registry, but not dynamic requirements validation. In [49] and [51], specifically evaluation aspects are covered. While, as pointed out earlier, these are not addressed here, a further integration of the aspect-specific composition through our $\cup^+$ with these concerns such as QoS in [49] is needed.

Applications are usually validated before deployment through testing and other means. With dynamically changing applications, shifting validation to runtime is important. *Runtime monitoring* of service processes is proposed by [7]. They use their own platform called *Dynamo* and their own annotation language. Monitoring rules are blended with a composite service process. Service composition is separated from rule blending, which is of our interest. However, they instrument the abstract service process before deployment. They assume stability of services in the abstract service process. On failure, redeployment is necessary. If a rule fails, the architect needs to change priorities or redeploy the process. Instrumentation is the most widely used monitoring mechanism [55]. The authors in [55] introduce an online monitoring approach for Web service requirements, where monitoring code is embedded inside the target code. Process instrumentation with monitoring rules before deployment is proposed through source code weaving in [6,9], in which a change in a monitoring code needs a redeployment of the whole process. An aspect-oriented extension for monitoring a BPEL process according to given QoS criteria to replace existing partner services is proposed in [39]. However, these approaches do not sufficiently address dynamic instrumentation of context constraints to a deployed service process for validation monitoring of requirements at process run-time. Context-based replanning [12] takes monitoring of context on board to determine replanning activities, thus moving the concern to implement self-* properties, which is beyond the scope here.

8 Conclusions

We have identified challenges that are not sufficiently addressed so far for the context modelling and management of dynamic Web service processes. An explicit formalisation of dynamic aspects relevant to the composition and execution of Web service processes, i.e. a conceptualisation of service contexts in the form of an ontology and an operationalisation through operators, has consequently been our aim. Defining dynamic requirements as context constraints demonstrates the tractability of the proposed context modelling approach. Context operationalisation with a Web service process in order to validate dynamic requirements at process runtime links into process context instrumentation and validation monitoring would utilise our context model. Our contribution includes:

- Firstly, a detailed context model ontology provides a shared conceptualisation of dynamic provisioning

and consumption aspects relevant to composition and execution of Web services.
– Secondly, an operator calculus for manipulation and composition of ontology-based context specifications.

While most context aspects are oriented towards services, our framework demonstrates the need to look at these from the perspective of processes as composed services. An application of the context model and our implementation of case study scenarios showed that our approach provides a practical solution.

We focused on *dynamic contextualization*, i.e., placing contextual aspects in a Web service process at process runtime. We discussed context modelling and context manipulation, composition and reasoning aspects on ontology-based context specifications in detail. Two case studies were used to illustrate dynamic contextualization for services and processes. A concern was to illustrate the suitability of an ontology framework to support rich knowledge structures such as interdependent context aspects and support them through an equally rich operator calculus.

There are several ways in which this context model can be utilised.

– Firstly, there is the context constraints generation, instrumentation and validation for dynamic service process. The context model presented can provide input and configure a monitoring solution [54].
– Secondly, for adaptation – statically or dynamically. We used the localisation example where the context model allows to capture the different context settings for consumer and provider [43].

In both cases, the operator framework for manipulation and composition of possible cross-category interdependent context aspects plays the central role.

The dynamic contextualization can be further reinforced by dynamic constraints selection, which is part of our future work. Some steps are documented in [5] where we have used CLiX (Constraint Language in XML) to provide dynamic context constraints processing. However, the full scope of the operator calculus is not yet supported dynamically. Furthermore, dynamic contextualization of context constraints resulting in dynamic recomposition is beyond our scope here.

Acknowledgements This work was supported, in part, by Science Foundation Ireland grant 07-RPF-CMSF429 (CASCAR project).

References

1. Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. Journal of IEEE Transactions on Dependable and Secure Computing **1**, 11–33 (2004).
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The description logic handbook - theory, implementation and applications. Cambridge University Press (2003).
3. Bandara, K.Y., Wang, M.X., Pahl, C.: Context modelling and constraints binding in web service business processes. In: Proceedings of the Workshop on Context-Aware Software Technology and Applications-CASTA. ESEC/FSE, ACM Press (2009)
4. Bandara, K.Y., Wang, M.X., Pahl, C.: Dynamic integration of context model constraints in web service processes. In: International Conference on Software Engineering. (2009).
5. Bandara, K.Y.: Ontology-based contextualization and context constraints management in Web service processes. PhD Thesis. Dublin City University. (2012).
6. Baresi, L., Ghezzi, C., Guinea, S.: Towards self-healing service compositions. In: Proceedings of the 1st Conference on the Principles of Software Engineering (2004).
7. Baresi, L., Guinea, S.: Towards dynamic monitoring of ws-bpel processes. pp. 269–282. Springer (2005)
8. Baresi, L., Guinea, S.: Self-supervising bpel processes. Journal of IEEE Transactions on Software Engineering **37**, 247–263 (2011).
9. Baresi, L., Guinea, S., Nano, O., Spanoudakis, G.: Comprehensive monitoring of bpel processes. Journal of IEEE Internet Computing **14**, 50–57 (2010).
10. Boukadi, K., Ghedira, C., Chaari, S., Vincent, L., Bataineh, E.: How to employ context, web service, and community in enterprise collaboration. In: Proceedings of the 8th International Conference on New Technologies in Distributed Systems. ACM (2008).
11. Bronsted, J., Hansen, K.M., Ingstrup, M.: Service Composition Issues in Pervasive Computing. IEEE Pervasive Computing 9(1), 60–72. (2010).
12. Bucchiarone, A., Pistore, M., Raik, H. and Kazhamiakin, R. Adaptation of service-based business processes by context-aware replanning. In Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on, pp. 1-8. (2011)
13. Chen, H., Perich, F., Finin, T., Joshi, A.: Soupa: Standard ontology for ubiquitous and pervasive applications. In: Proc. International Conference on Mobile and Ubiquitous Systems: Networking and Services. (2004).
14. Chung, L., Prado, L., Julio, C.: On non-functional requirements in software engineering. In: A.T. Borgida, V.K. Chaudhri, P. Giorgini, E.S. Yu (eds.) Conceptual modelling : Foundations and Applications, pp. 363–379. Springer (2009).
15. Cohen, W.W., Borgida, A., Hirsh, H.: Computing least common subsumers in description logics. In: Proc. 10th National Conference on Artificial Intelligence. (1992)
16. Coutaz, J., Crowley, J., Dobson, S., Garlan, D.: Context is key. Journal of Communications of the ACM **48**, 49–53. (2005).
17. Daconta, M., Obrst, L., Smith, K.: The semantic web: A guide to the future of xml, web services, and knowledge management. Wiley (2003)
18. Dey, A.: Providing architectural support for building context-aware applications. Ph.D. thesis, Georgia Institute of Technology. (2000).
19. Doulkeridis, C., Loutas, N., Vazirgiannis, M.: A system architecture for context aware service discovery. Journal of Electronic Notes in Theoretical Computer Science pp. 101–116 (2006).

20. Dustdar, S., Papazoglou, M.: Services and service composition - an introduction. Journal of Information Technology **50**, 86–92 (2008).
21. Euzenat, J., Pierson, J., Ramparany, F.: Dynamic context management for pervasive applications. Journal of The Knowledge Engineering Review **23**, 21–49 (2008).
22. Farrar, S., Langendoen, D.T.: An owl-dl implementation of gold- an ontology for the semantic web. Journal of Linguistic modelling of Information and Markup Languages **40**, 45–66 (2010).
23. Fujii, K., Suda, T.: Semantics-based context-aware dynamic service composition. ACM Transactions on Autonomous and Adaptive Systems **4**, 1–31 (2009).
24. Goslar, K., Schill, A.: modelling contextual information using active data structures. In: Proceedings of the EDBT Workshops, *Lecture Notes in Computer Science*, vol. 3268. Springer (2004).
25. Harry, C., J., A.: The soupa ontology for pervasive computing. In: The Volume on Ontologies for Agent Systems, pp. 233–258. Birkhauser Publishing Ltd. (2004).
26. Hasselbring, W., Reussner, R.: Toward trustworthy software systems. IEEE Computer **39**, 91–92 (2006).
27. Heravizadeh, M., Mendling, J., Rosemann, M.: Dimensions of business process quality. In: Proceedings of the 6th International Conference on Business Process Management Workshop, pp. 80–91. Springer (2008)
28. Hong, M., Cho, D.: Ontology context model for context aware learning service in ubiquitous learning environments. Intl Journal of Computers pp. 193–200 (2008).
29. Horrocks, I., Patel-Schneider, F.: Reducing owl entailment to description logic satisfiability. The Semantic Web - ISWC 2003, Lecture Notes in Computer Science **2870**, 17–29 (2003).
30. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: Swrl: A semantic web rule language combining owl and ruleml. W3C Submission (2004).
31. Kapitsaki, G., Kateros, D., Prezerakos, G., Venierris, I.: Model-driven development of composite context-aware web applications. Journal of Information and Software Technology **51**, 1244–1260 (2009).
32. Kozen, D., Tiuryn, J.: Logics of programs. In: Handbook of Theoretical Computer Science, vol. B: Formal Models and Sematics, pp. 789–840 (1990).
33. Lee, K.C., Kim, J., Lee, J., Lee, K.M.: Implementation of ontology based context aware framework for ubiquitous environments. In: Proc. International Conference on Multimedia and Ubiquitous Engineering (2007).
34. Maamar, Z., Benslimane, D., Narendra, N.: What can context do for web services? Communications of the ACM **49**, 98–103 (2006).
35. Marquezan, C.C., Metzger, A., Pohl, K., Engen, V., Boniface, M., Phillips, S.C. and Zlatev, Z. Adaptive future internet applications: Opportunities and challenges for adaptive web services technology. Adaptive Web Services for Modular and Reusable Software Development, G. Ortiz and J. Cubo, Eds. IGI Global (2012).
36. Martin, D.: Putting web services in context. Electronic Notes in Theoretical Computer Science **146**, 3–16 (2006).
37. Medjahed, B., Atif, Y.: Context-based matching for web service composition. Journal of Distributed and Parallel Databases **21**, 5–37 (2007).
38. Medjahed, B., Bouguettaya, A.: A dynamic foundation architecture for semantic web services. Journal of Distributed and Parallel Databases **17**, 179–206 (2005).
39. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for ws-bpel. In: Proceeding of the 17th International Conference on World Wide Web, WWW '08. ACM (2008).
40. Mrissa, M., Thiran, P., Ghedira, C., Benslimane, D., Maamar, Z.: Using context to enable semantic mediation in web service communities. In: Proc. 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation. ACM (2008).
41. Pahl, C., Bandara, K.Y., Wang, M.X.: Context constraint integration and validation. In: Q.Z. Sheng, J. Yu, S. Dustdar (eds.) Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, 1st edn., pp. 81–105. Chapman & Hall/CRC (2010).
42. Pahl, C., Giesecke, S., Hasselbring, W.: Ontology-based modelling of architectural styles. Information and Software Technology **51**, 1739–1749 (2009).
43. Pahl, C. Cloud Service Localisation. European Conference on Service-Oriented and Cloud Computing ESOCC 2012. Springer LNCS. (2012).
44. Priestley, M.: Practical object-oriented design with uml. McGraw Hill Higher Education (2003).
45. Raik, H., Bucchiarone, A., Khurshid, N., Marconi, A. and Pistore, M. Astro-captevo: Dynamic context-aware adaptation for service-based systems. In Services (SERVICES), 2012 IEEE Eighth World Congress on, pp. 385-392. IEEE. (2012).
46. Romero, D., Rouvoy, R., Seinturier, L., Chabridon, S., Conan, D., and Pessemier, N. Enabling context-aware web services: a middleware approach for ubiquitous environments. Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, pp. 113-135. (2010).
47. Rosemann, M., Recker, J.C., Flender, C.: Contextualisation of business processes. Journal of Business Process Integration and Management **3**, 47–60 (2008).
48. Roy, N., Gu, T., Das, S.: Supporting pervasive computing applications with active context fusion and semantic context delivery. Journal of Pervasive and Mobile Computing **6**, 21–42 (2010).
49. Sathya, M., Swarnamugi, M., Dhavachelvan, P., Sureshkumar, G.: Evaluation of qos based web-service selection techniques for service composition. Intl Journal of Software Engineering, 1(5):73-90. (2010).
50. Sheshagiri, M., Sadeh, N., Gandon, F.: Using semantic web services for context aware mobile applications. Proceedings of the MobiSys 2004 Workshop on Context-Awareness (2004).
51. Silva, E., Pires, L.F., van Sinderen, M.: A framework for the evaluation of semantics-based service composition approaches. Proc. 7th IEEE European Conference on Web Services ECOWS. (2009).
52. Truong, H., Dustdar, S.: A survey on context-aware web service systems. International Journal of Web Information Systems **5**, 5–31 (2009).
53. Vladimir, S., Miroslaw, M.: Addressing dependability throughout the soa life cycle. IEEE Transactions on Services Computing **4**, 85–95 (2011).
54. Wang, M.X., Bandara, K.Y. and Pahl, C.: Process as a Service - Distributed Multi-tenant Policy-based Process Runtime Governance IEEE International Conference on Services Computing SCC 2010. (2010).
55. Wang, Q., Shao, J., Deng, F., Liu, Y., Li, M., Han, J., Mei, H.: An online monitoring approach for web service requirements. IEEE Transactions on Services Computing **2**, 338–351 (2009).
56. Wang, X., Zhang, D.Q., Gu, T., Pung, H.: Ontology based context modelling and reasoning using owl. In: Proc. of the 2nd Annual Conference on Pervasive Computing and Communications Workshops. IEEE. (2004).